References

[1] Charles Way Hun Fung, Antˆonio Gortan, and Walter Godoy Junior, “A Review Study on Image Digital Watermarking,” in *The Tenth International Conference on Networks*, 2011, pp. 24–28.

[2] M. L. Miller, I. J. Cox, J.-P. M. Linnartz, and T. Kalker, “A review of watermarking principles and practices,” *Digit. Signal Process. Multimed. Syst.*, pp. 461–485, 1999.

[3] H. Tao, L. Chongmin, J. M. Zain, and A. N. Abdalla, “Robust image watermarking theories and techniques: A review,” *J. Appl. Res. Technol.*, vol. 12, no. 1, pp. 122–138, 2014.

[4] Joachim von zur Gathen and El-Gayyar, “Watermarking Techniques Spatial Domain Digital Rights Seminar copyright.”

[5] C. Nafornita, A. Isar, and M. Borda, “Improved Pixel-Wise Masking for Image Watermarking,” in *Multimedia Content Representation, Classification and Security*, vol. 4105, B. Gunsel, A. K. Jain, A. M. Tekalp, and B. Sankur, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 90–97.

[6] C. Nafornita, A. Isar, and M. Borda, “Pixel-wise masking for watermarking using local standard deviation and wavelet compression,” *Sci. Bull. Politeh. Univ Timisoara Trans Electron. Telecommun.*, vol. 51, no. 65, pp. 146–151, 2006.

[7] H.-J. Wang, P.-C. Su, and C.-C. J. Kuo, “Wavelet-based digital image watermarking,” *Opt. Express*, vol. 3, no. 12, pp. 491–496, 1998.

[8] Mei Jiansheng, Li Sukang, and Tan Xiaomei, “A Digital Watermarking Algorithm Based On DCT and DWT,” *Int. Symp. Web Inf. Syst. Appl.*, pp. 22–24, May 2009.

[9] K. Loukhaoukha, “Security of ownership watermarking of digital images based on singular value decomposition,” *J. Electron. Imaging*, vol. 19, no. 1, p. 013007, Jan. 2010.

[10] K. Loukhaoukha, J.-Y. Chouinard, and M. H. Taieb, “Multi-Objective Genetic Algorithm Optimization for Image Watermarking Based on Singular Value Decomposition and Lifting Wavelet Transform,” in *Image and Signal Processing*, vol. 6134, A. Elmoataz, O. Lezoray, F. Nouboud, D. Mammass, and J. Meunier, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 394–403.

[11] K. Loukhaoukha and J.-Y. Chouinard, “Hybrid watermarking algorithm based on SVD and lifting wavelet transform for ownership verification,” 2009, pp. 177–182.

[12] D.-C. Wu and W.-H. Tsai, “A steganographic method for images by pixel-value differencing,” *Pattern Recognit. Lett.*, vol. 24, no. 9–10, pp. 1613–1626, Jun. 2003.

[13] K.-C. Chang, C.-P. Chang, P. S. Huang, and T.-M. Tu, “A novel image steganographic method using tri-way pixel-value differencing,” *J. Multimed.*, vol. 3, no. 2, pp. 37–44, 2008.

[14] X. Wu, J. Hu, Z. Gu, and J. Huang, “A secure semi-fragile watermarking for image authentication based on integer wavelet transform with parameters,” in *Proceedings of the 2005 Australasian workshop on Grid computing and e-research-Volume 44*, 2005, pp. 75–80.

[15] P. Campisi, D. Kundur, and A. Neri, “Robust Digital Watermarking in the Ridgelet Domain,” *IEEE Signal Process. Lett.*, vol. 11, no. 10, pp. 826–830, Oct. 2004.

[16] K. Manashee and T. Themrichon, “A Comparative Study of Steganography Algorithms of Spatial and Transform Domain - ncit175194.pdf,” *Int. J. Comput. Appl.*, 2015.

[17] M. Kim, D. Li, and S. Hong, “A Robust Digital Watermarking Technique for Image Contents based on DWT-DFRNT Multiple Transform Method,” *Int. J. Multimed. Ubiquitous Eng.*, vol. 9, no. 1, pp. 369–378, Jan. 2014.

[18] Z. Yuefeng and L. Li, “DIGITAL IMAGE WATERMARKING ALGORITHMS BASED ON DUAL TRANSFORM DOMAIN AND SELF-RECOVERY,” *Int. J. Smart Sens. Intell. Syst.*, vol. 8, no. 1, 2015.

[19] Z. Dawei, C. Guanrong, and L. Wenbo, “A chaos-based robust wavelet-domain watermarking algorithm,” *Chaos Solitons Fractals*, vol. 22, no. 1, pp. 47–54, Oct. 2004.

[20] N. Nikolaidis and I. Pitas, “Robust image watermarking in the spatial domain,” *Signal Process.*, vol. 66, no. 3, pp. 385–403, 1998.

[21] F. Sebé, J. Domingo-Ferrer, and J. Herrera, “Spatial-domain image watermarking robust against compression, filtering, cropping, and scaling,” in *Information Security*, Springer, 2000, pp. 44–53.

[22] E. Praun, H. Hoppe, and A. Finkelstein, “Robust mesh watermarking,” in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999, pp. 49–56.

[23] A. Bamatraf, R. Ibrahim, and M. N. B. M. Salleh, “Digital watermarking algorithm using LSB,” 2010, pp. 155–159.

[24] Farid Ahmed and Ira S. Moskowitz, “Correlation-based watermarking method for image authentication applications,” *Opt. Eng.*, vol. 43, no. 8, Feb. 2004.

[25] R. L. De Queiroz, “Processing JPEG-compressed images and documents,” *Image Process. IEEE Trans. On*, vol. 7, no. 12, pp. 1661–1672, 1998.

[26] G. K. Wallace, “The JPEG still picture compression standard,” *IEEE Trans. Consum. Electron.*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[27] A. N. Skodras, C. A. Christopoulos, and T. Ebrahimi, “JPEG2000: The upcoming still image compression standard,” *Pattern Recognit. Lett.*, vol. 22, no. 12, pp. 1337–1345, 2001.

[28] H. Wood, “Invisible Digital Watermarking the Spatial and DCT Domains for Color Images,” *Adams State Coll. Alamosa Colo.*

[29] X. Qi, “An Efficient Wavelet-based Watermarking Algorithm,” in *proceedings of Hawaii International Conference on Computer Sciences*, 2002, pp. 383–388.

[30] S. Zhang and K. Yoshino, “DWT-Based Watermarking Using QR Code,” 2008.

[31] C. Nafornita, A. Isar, and M. Borda, “Image Watermarking Based on the Discrete Wavelet Transform Statistical Characteristics,” 2005, pp. 943–946.

[32] C. Nafornita, A. Isar, and M. Kovaci, “Increasing watermarking robustness using turbo codes,” 2009, pp. 113–118.

[33] J. Ayubi, S. Mohanna, F. Mohanna, and M. Rezaei, “A chaos based blind digital image watermarking in the wavelet transform domain,” *Int. J. Comput. Sci. Issues*, vol. 8, no. 4, 2011.

[34] M. N. Do and M. Vetterli, “The finite ridgelet transform for image representation,” *IEEE Trans. Image Process.*, vol. 12, no. 1, pp. 16–28, 2003.

[35] C. Harris and M. Stephens, “A Combined Corner and Edge Detector,” 1988, p. 23.1-23.6.

[36] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[37] F. Jin and D. Feng, “Image Registration Algorithm Using Mexican Hat Function-Based Operator and Grouped Feature Matching Strategy,” *PLoS ONE*, vol. 9, no. 4, p. e95576, Apr. 2014.

[38] D. K. Park, Y. S. Jeon, and C. S. Won, “Efficient use of local edge histogram descriptor,” in *Proceedings of the 2000 ACM workshops on Multimedia*, 2000, pp. 51–54.

[39] C. S. Won, D. K. Park, and S.-J. Park, “Efficient use of MPEG-7 edge histogram descriptor,” *Etri J.*, vol. 24, no. 1, pp. 23–30, 2002.

[40] J. Fridrich and M. Goljan, “Digital image steganography using stochastic modulation,” in *Electronic Imaging 2003*, 2003, pp. 191–202.

[41] B. Dirk and A. RWTH, “stochastic modulation.”

[42] J. J. Harmsen, K. D. Bowers, and W. A. Pearlman, “Fast additive noise steganalysis,” in *Electronic Imaging 2004*, 2004, pp. 489–495.

[43] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck, *Discrete time Signal Processing*, 2nd Edition.

[44] V. Senthooran and L. Ranathunga, “DCT coefficient dependent quantization table modification steganographic algorithm,” in *Networks & Soft Computing (ICNSC), 2014 First International Conference on*, 2014, pp. 432–436.

[45] A. Zigomitros and C. Patsakis, “Cross format embedding of metadata in images using QR codes,” in *Intelligent Interactive Multimedia Systems and Services*, Springer, 2011, pp. 113–121.

[46] H.-Y. Lee, I. K. Kang, H.-K. Lee, and Y.-H. Suh, “Evaluation of feature extraction techniques for robust watermarking,” in *International Workshop on Digital Watermarking*, 2005, pp. 418–431.

[47] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, 2004.

[48] Z. Wang, E. P. Simoncelli, and A. C. Bovik, “Multiscale structural similarity for image quality assessment,” in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, 2003, vol. 2, pp. 1398–1402.

[49] C. Li and A. C. Bovik, “Three-component weighted structural similarity index,” in *IS&T/SPIE Electronic Imaging*, 2009, p. 72420Q–72420Q.

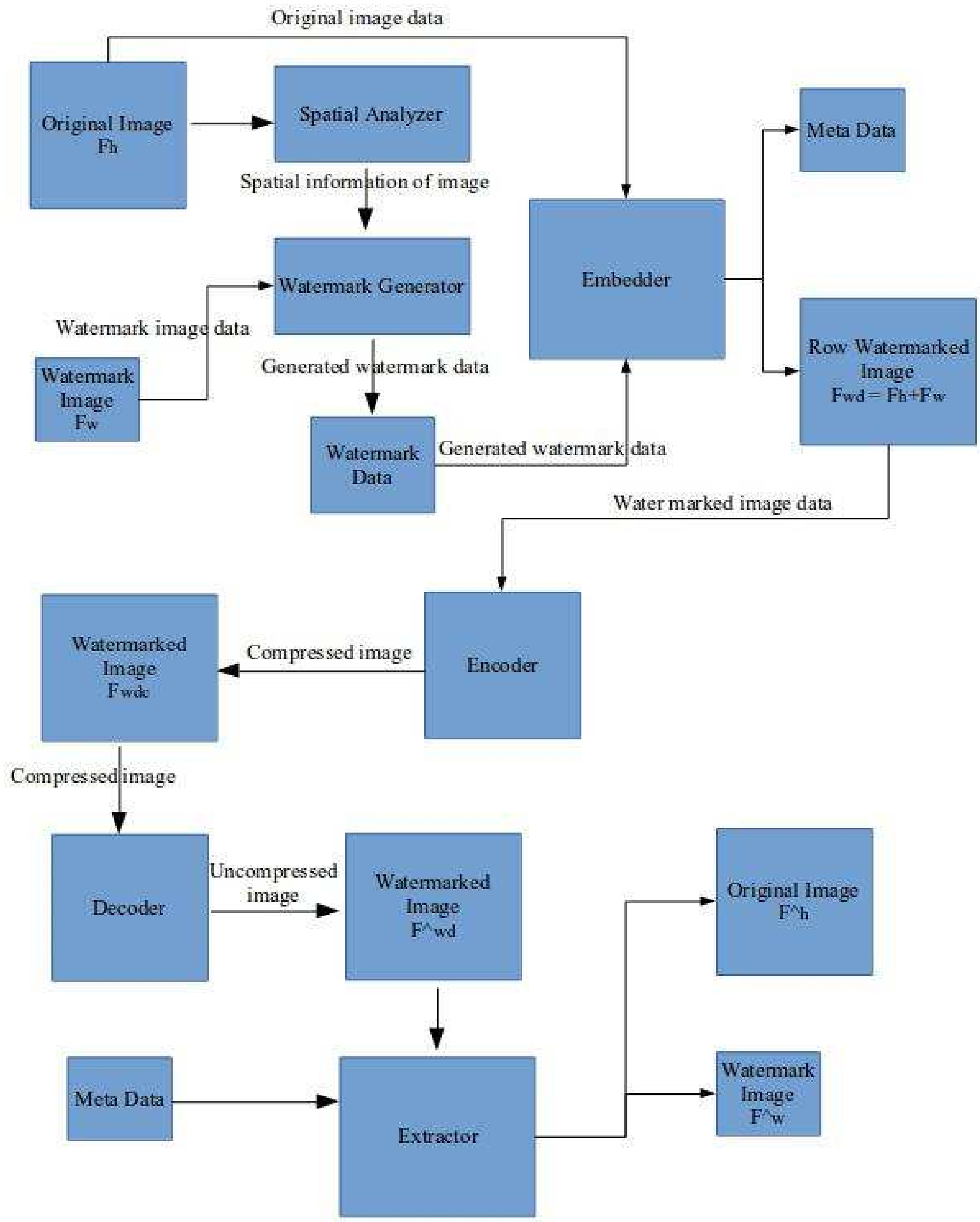
[50] Lin Zhang, Lei Zhang, Xuanqin Mou, and Daivd Zhang, “FSIM: A Feature Similarity Index for Image Quality Assessment.”

[51] T. Shohdohji, Y. Hoshino, and N. Kutsuwada, “Optimization of quantization table based on visual characteristics in DCT image coding,” *Comput. Math. Appl.*, vol. 37, no. 11–12, pp. 225–232, Jun. 1999.

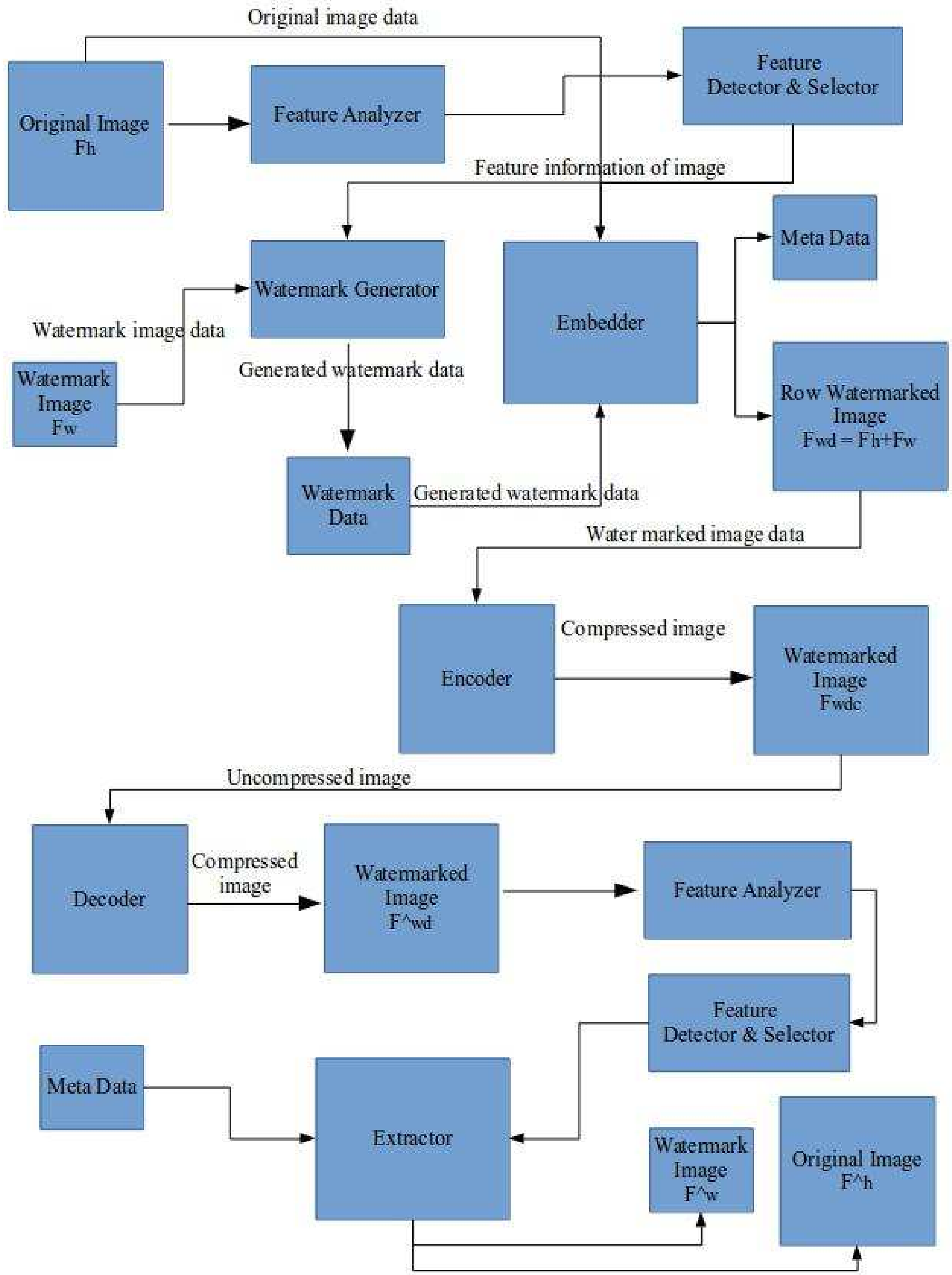
[52] Q. Li, C. Yuan, and Y. Z. Zhong, “Adaptive DWT-SVD Domain Image Watermarking Using Human Visual Model,” in *the 9th International Conference on*

Advanced Communication Technology, 2007, vol. 3, pp. 1947–1951.

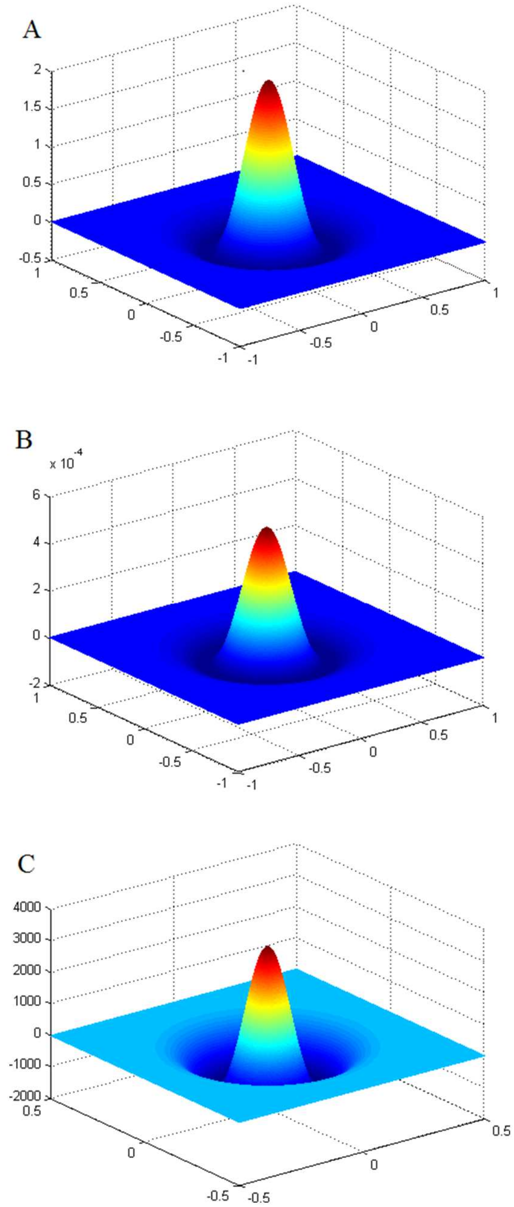
Appendix A (Methodology)



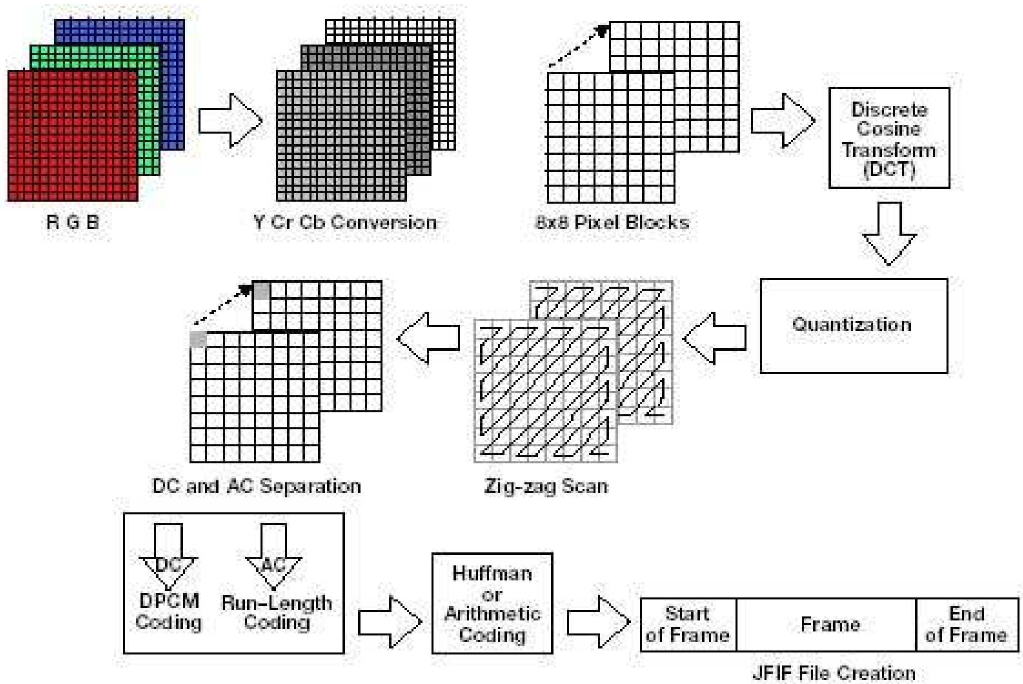
Appendix A, Figure 1: High level design diagram of pixel base watermarking system



Appendix A, Figure 2: High level design diagram of feature base watermarking system



Appendix A, Figure 3: Mexican hat operator with different sigma values. (Source: http://journals.plos.org)



Appendix A, Figure 3: Encoder module. *(Source: http://www.eetimes.com/)*

Appendix B (Experimental Design)

Dataset Used in Research

Name: lena1.bmp

Format: bit map

Dimension: 640x640 Px

Size: 1.2Mb

Name: lena2.jpg

Format: jpg

Dimension: 512x512 Px

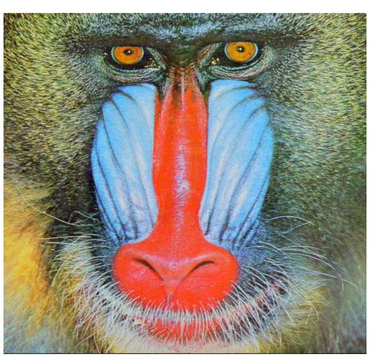
Size: 94.4 Kb

Name: lena3.png

Format: png

Dimension: 512x512 Px

Size: 473.8 Kb



Name: monky.png

Format: png

Dimension: 512x512 Px

Size: 626.9 Kb

Name: veg.jpg

Format: jpg

Dimension: 512x512 Px

Size: 48.5 Kb

Experimental results of pixel based watermark

Experiment 1: Embed using random insertion

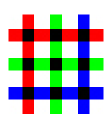
Inputs	Outputs
Base method : Pixel base Technique : Random insertion Original image : lena.jpg Watermark : Watermark 4	Watermarked image : result_lena.jpg Metadata file : result_lena.csv

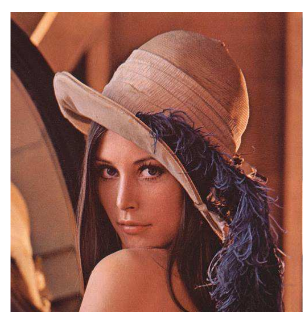
Compression parameters:

Block size: 8x8

Quality: 70%

JSAMPROW row pointer size: 1





Original image	Watermark image	Watermarked image
Dimension: 512x512	Dimension: 48x48	Dimension: 512x512
Size: 94.4 Kb	Size: 22.8 Kb	Size: 37.9 Kb

Experiment 2: Embed using less sensitive points of human vision

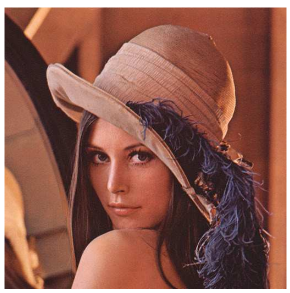
Inputs	Outputs
Base method : Pixel base Technique : less sensitive points of human vision Original image : lena.jpg Watermark : Watermark 4	Watermarked image : result_lena.jpg Metadata file : result_lena.csv

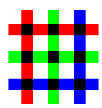
Compression parameters:

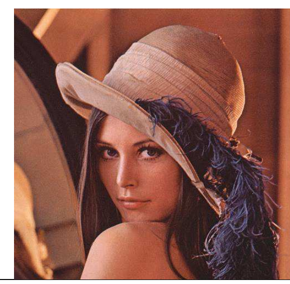
Block size: 8x8

Quality: 70%

JSAMPROW row pointer size: 1







Original image Dimension: 512x512 Size: 94.4 Kb	Watermark image Dimension: 48x48 Size: 22.8 Kb	Watermarked image Dimension: 512x512 Size: 37.9 Kb

Experiment 3: Embed using LSB of macro-block

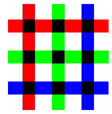
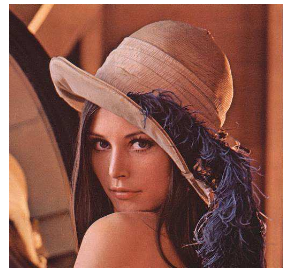
Inputs	Outputs
Base method : Pixel base Technique : LSB of macro-block Original image : lena.jpg Watermark : Watermark 4	Watermarked image : result_lena.jpg Metadata file : result_lena.csv

Compression parameters:

Block size: 8x8

Quality: 70%

JSAMPROW row pointer size: 1

Original image Dimension: 512x512 Size: 37.9 Kb	Watermark image Dimension: 48x48 Size: 22.8 Kb	Watermarked image Dimension: 512x512 Size: 45.4 Kb

Experimental results of pixel based watermark extraction methods

Experiment 4: Extraction using common algorithm

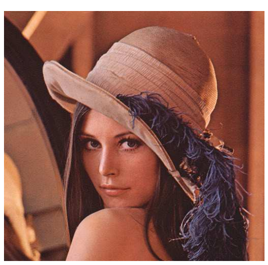
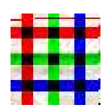
Inputs	Outputs
Base method : Pixel base Technique : common Original image : result_lena.jpg Metadata file : result_lena.csv	Extracted watermark: wtm.jpg

Compression parameters:

Block size: 8x8

Quality: 70%

JSAMPROW row pointer size: 1

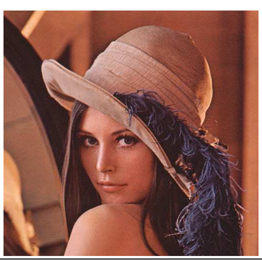
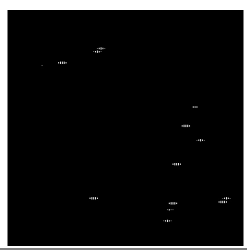
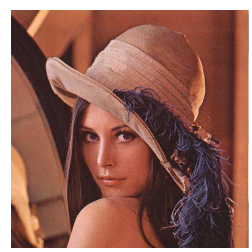



Watermarked image Dimension: 512x512 Size: 45.4 Kb	Extracted watermark Dimension: 48x48 Size: 1.9 Kb

Experimental results of feature based watermark extraction methods

Experiment 5: Embed using Harris corner detector

Inputs	Base method : Feature base Possession : Around single corner Original image : lena.jpg Watermark : Watermark 4
Outputs	Watermarked image : result_lena.jpg Metadata file : result_lena.csv
Settings	Feature detector: Harris operator Smoothing: Gaussian filter Kernel: 5x5 Sigma: 1 Threshold: 120
Compression parameters	Block size: 8x8 Quality: 70% JSAMPROW row pointer size: 1

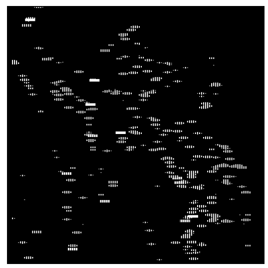




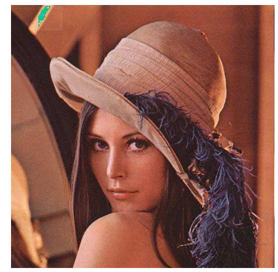
Original image Dimension: 512x512 Size: 37.9 Kb	Binary corner image Dimension: 512x512 Size: 16.2 Kb	Watermarked image Dimension: 512x512 Size: 47.4 Kb

Experiment 6: Embed using novel corner detector

Inputs	Base method : Feature base Position : Around single corner Original image : lena.jpg Watermark : Watermark 4
Outputs	Watermarked image : result_lena.jpg Metadata file : result_lena.csv
Settings	Feature detector: Novel operator Smoothing: LoG

	Kernel: 5x5 Sigma: 1 Threshold: 120
Compression parameters	Block size: 8x8 Quality: 70% JSAMPROW row pointer size: 1

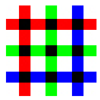
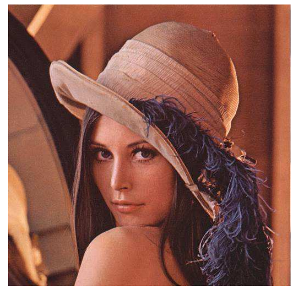
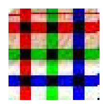
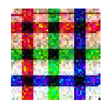
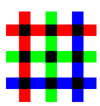
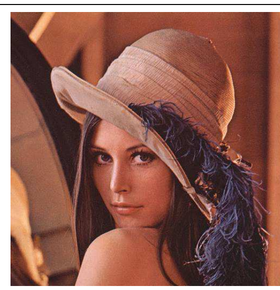
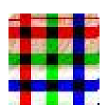
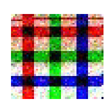


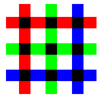
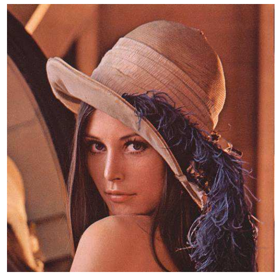
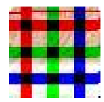
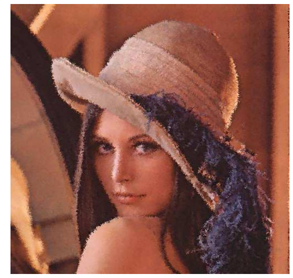
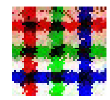
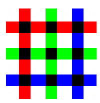
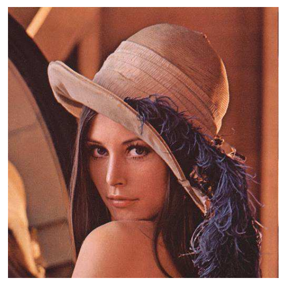
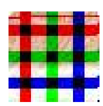
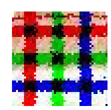


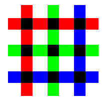
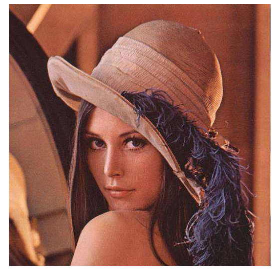
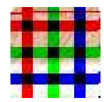
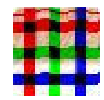
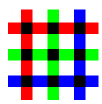
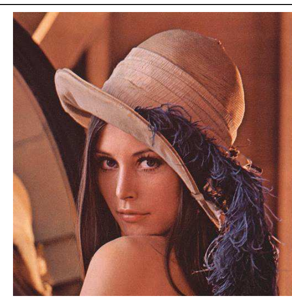
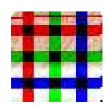
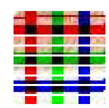
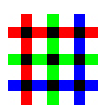
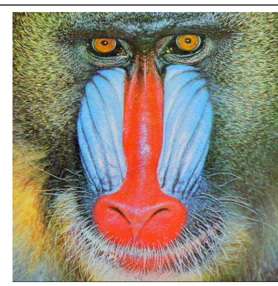
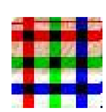
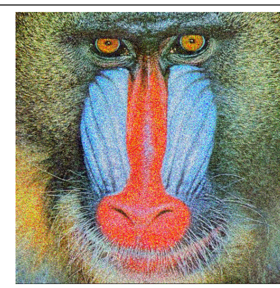
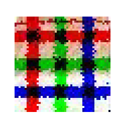
Original image Dimension: 512x512 Size: 37.9 Kb	Binary corner image Dimension: 512x512 Size: 18.3 Kb	Watermarked image Dimension: 512x512 Size: 46.2 Kb

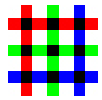
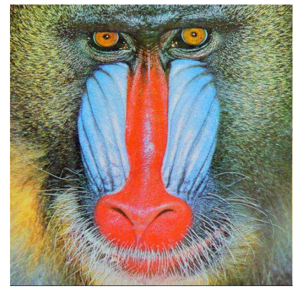
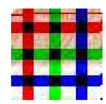
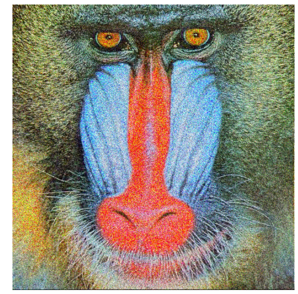
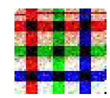
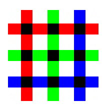
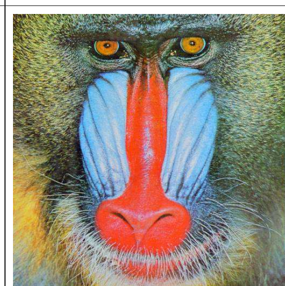
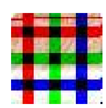
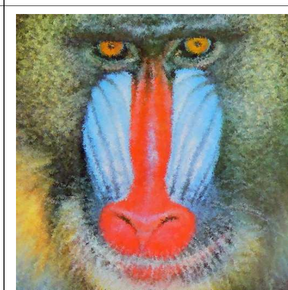
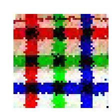
Appendix C (Evaluation)

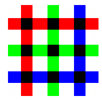
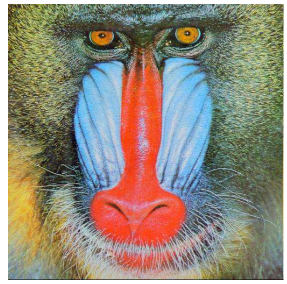
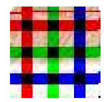
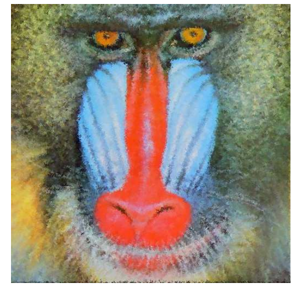
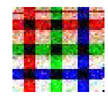
Evaluate of the Robustness

Settings & Attack	Watermark	Watermarked Image	Watermarked Image After Attack
Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: single corner **HSV noise** [Holdness: 3] [Hue:72] [Saturation:146] [Value: 94]		 	 
Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1 Threshold: 125 Position: single corner **HSV noise** [Holdness: 3] [Hue:72] [Saturation:146] [Value: 94]		 	 

Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1.5 Threshold: 125 Position: single corner **Random Pik.** [Random seeds: 1452988117] [Randomization: 42] [Repeat: 5]		 	 
Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1.5 Threshold: 125 Position: single corner **Random Pik.** [Random seeds: 1452988117] [Randomization: 42] [Repeat: 5]		 	 

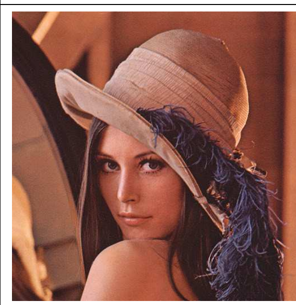
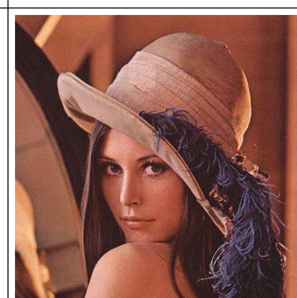
Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1.5 Threshold: 125 Position: single corner **Rotation in 45 degree**		 	 
Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1.5 Threshold: 125 Position: single corner **Scaling** watermarked image: 512x512 scaled to: 256x256		 	 
Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: single corner **HSV noise** [Holdness: 3] [Hue:72] [Saturation:146] [Value: 94]		 	 

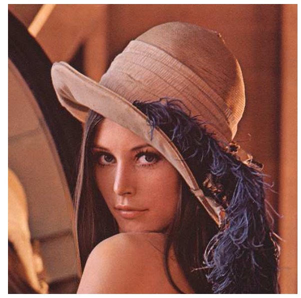
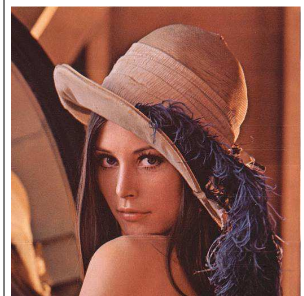
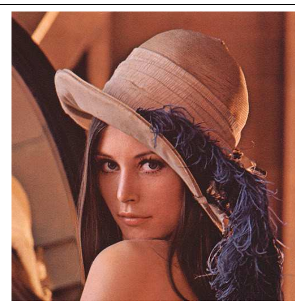
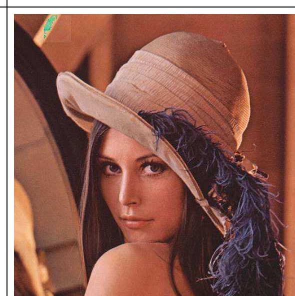
Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1 Threshold: 125 Position: single corner **HSV noise** [Holdness: 3] [Hue:72] [Saturation:146] [Value: 94]		 	 
Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: single corner **Random Pik.** [Random seeds: 1452988117] [Randomization: 42] [Repeat: 5]		 	 

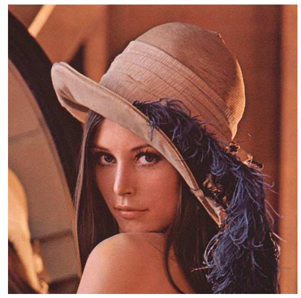
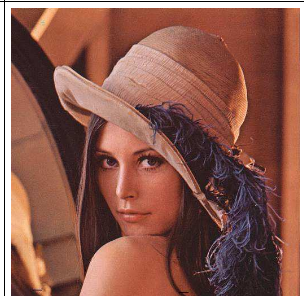
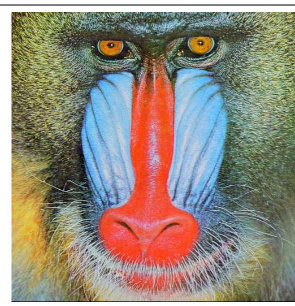
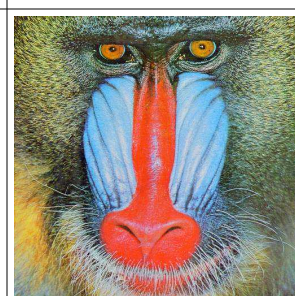
Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1 Threshold: 125 Position: single corner **Random Pik.** [Random seeds: 1452988117] [Randomization: 42] [Repeat: 5]		 	 

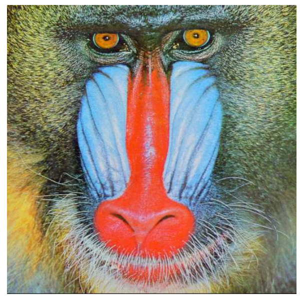
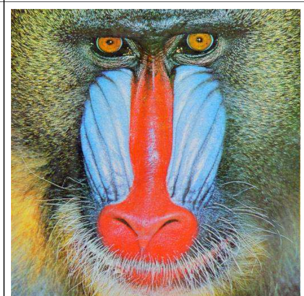
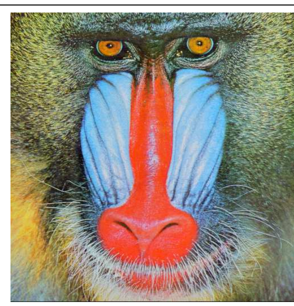
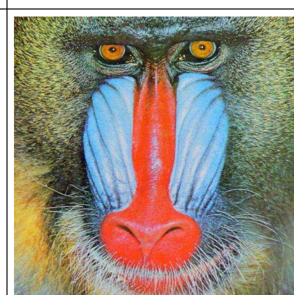
Appendix C, Table 1: Summary of evaluation results for the robustness

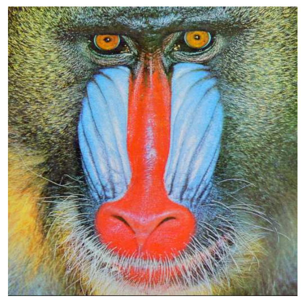
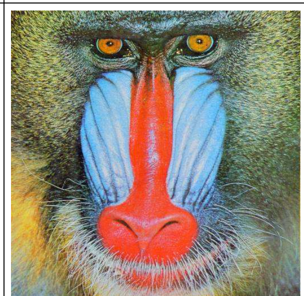
Evaluate of the Fidelity

Image Detail & Settings	**Evaluation Results**	**Original Image**	**Watermarked Image**
Image: lena.jpg (512x512, 94.4 Kb) **Settings** Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: single corner	**Evaluation Method 1: MSE** Resulting Value: 102.03 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 39.28 Conclusion: Good fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.69 Chanel[1]: 0.75 Chanel[2]: 0.74 Mean: 0.72 Conclusion: Good		

	Fidelity		
Image: lena.jpg (512x512, 94.4 Kb) **Settings** Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: multiple corners	**Evaluation Method 1: MSE** Resulting Value: 113.03 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 41.28 Conclusion: Good fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.69 Chanel[1]: 0.75 Chanel[2]: 0.74 Mean: 0.72 Conclusion: Good Fidelity		
Image: lena.jpg (512x512, 94.4 Kb) **Settings** Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1 Threshold: 125 Position: single corner	**Evaluation Method 1: MSE** Resulting Value: 76.03 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 44.36 Conclusion: Very good fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.79 Chanel[1]: 0.85		

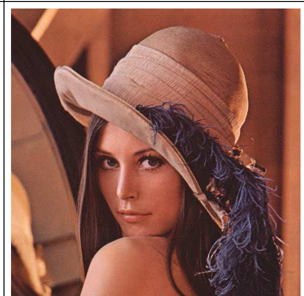
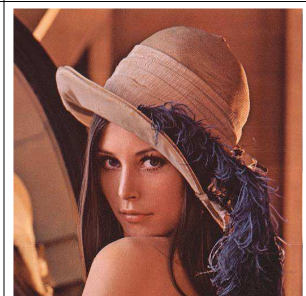
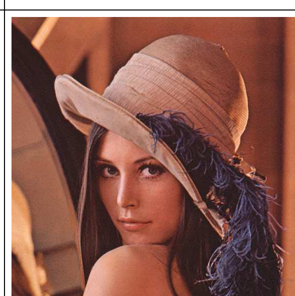
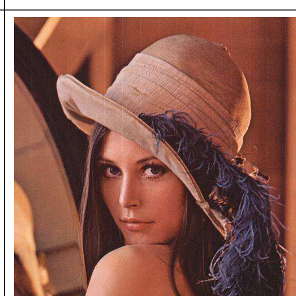
	Chanel[2]: 0.84 Mean: 0.82 Conclusion: Very good Fidelity		
Image: lena.jpg (512x512, 94.4 Kb) **Settings** Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1 Threshold: 125 Position: multiple corners	**Evaluation Method 1: MSE** Resulting Value: 54.44 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 45.51 Conclusion: Very good fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.79 Chanel[1]: 0.85 Chanel[2]: 0.84 Mean: 0.82 Conclusion: Very good Fidelity		
Image: monky.png (512x512, *626.9 Kb*) **Settings** Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: single corner	**Evaluation Method 1: MSE** Resulting Value: 202.03 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 38.80 Conclusion: Good fidelity **Evaluation Method 3: SSIM**		

	Resulting Value: Chanel[0]: 0.69 Chanel[1]: 0.75 Chanel[2]: 0.74 Mean: 0.72 Conclusion: Good Fidelity		
Image: monky.png (512x512, *626.9* Kb) **Settings** Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: multiple corners	**Evaluation Method 1: MSE** Resulting Value: 203.03 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 38.28 Conclusion: Good fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.69 Chanel[1]: 0.75 Chanel[2]: 0.74 Mean: 0.72 Conclusion: Good Fidelity		
Image: monky.png (512x512, *626.9* Kb) **Settings** Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1 Threshold: 125	**Evaluation Method 1: MSE** Resulting Value: 146.03 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 44.06 Conclusion: Very good fidelity		

Position: single corner	**Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.79 Chanel[1]: 0.85 Chanel[2]: 0.84 Mean: 0.82 Conclusion: Very good Fidelity		
Image: monky.png (512x512, *626.9* Kb) **Settings** Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1 Threshold: 125 Position: multiple corners	**Evaluation Method 1: MSE** Resulting Value: 54.44 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 45.01 Conclusion: Very good fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.80 Chanel[1]: 0.85 Chanel[2]: 0.85 Mean: 0.84 Conclusion: Very good Fidelity		

Appendix C, Table 2: Summary of evaluation results for the fidelity

Evaluate of the Capacity

Image Detail & Settings	Evaluation Results (Statistically)	Evaluation Results (Experimentally) *Original Image*	*Evaluation Results (Experimentally) Watermarked Image*
Image: lena.jpg (512x512, 94.4 Kb) **Watermark object:** Name: watermark 4.jpg Dimension: 48x48 px, Size: 22.8 Kb **Settings** Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: multiple corner	**Evaluation Method 1: MSE** Resulting Value: 103.03 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 41.28 Conclusion: Good fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.69 Chanel[1]: 0.75 Chanel[2]: 0.74 Mean: 0.72 Conclusion: Good Fidelity		
Image: lena.jpg (512x512, 94.4 Kb) **Watermark object:** Name: watermark 4.jpg Dimension: 64x64 px, Size: 27.6 Kb	**Evaluation Method 1: MSE** Resulting Value: 183.04 Conclusion: Fair fidelity **Evaluation Method 2: PSNR** Resulting Value: 37.44 Conclusion: Fair fidelity **Evaluation Method 3: SSIM**		

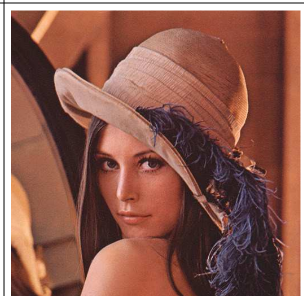
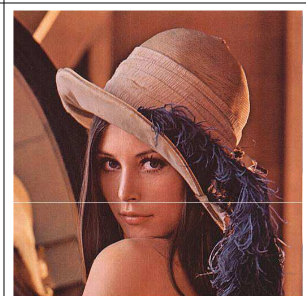
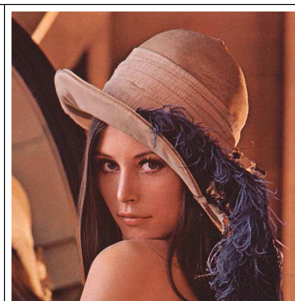
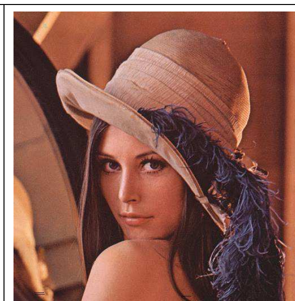
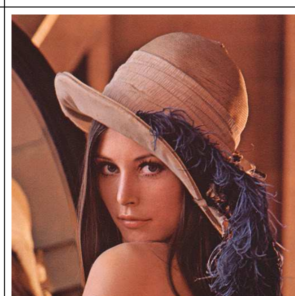
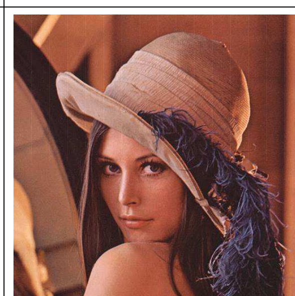
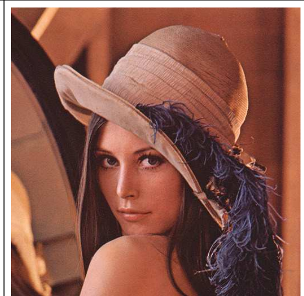
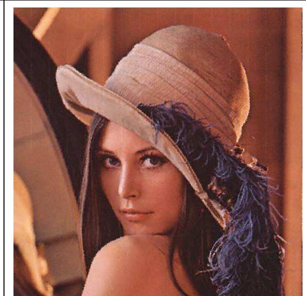
Settings Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: multiple corners	Resulting Value: Chanel[0]: 0.69 Chanel[1]: 0.75 Chanel[2]: 0.74 Mean: 0.72 Conclusion: Good Fidelity		
Image: lena.jpg (512x512, 94.4 Kb) **Watermark object:** Name: watermark 4.jpg Dimension: 128x128 px, Size: 29.8 Kb **Settings** Feature detector: Harris operator Smoothing: Gaussian Kernel: 5x5 Sigma: 1 Threshold: 125 Position: multiple corners	**Evaluation Method 1: MSE** Resulting Value: 344.05 Conclusion: Very bad fidelity **Evaluation Method 2: PSNR** Resulting Value: 23.82 Conclusion: Very bad fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.59 Chanel[1]: 0.64 Chanel[2]: 0.66 Mean: 0.63 Conclusion: Bad Fidelity		

Image: lena.jpg (512x512, 94.4 Kb) **Watermark object:** Name: watermark 4.jpg Dimension: 48x48 px, Size: 22.8 Kb **Settings** Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1 Threshold: 125 Position: multiple corners	**Evaluation Method 1: MSE** Resulting Value: 54.44 Conclusion: Very good fidelity **Evaluation Method 2: PSNR** Resulting Value: 45.51 Conclusion: Very good fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.79 Chanel[1]: 0.85 Chanel[2]: 0.84 Mean: 0.82 Conclusion: Very good Fidelity		
Image: lena.jpg (512x512, 94.4 Kb) **Watermark object:** Name: watermark 4.jpg Dimension: 64x64 px, Size: 27.6 Kb **Settings** Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1	**Evaluation Method 1: MSE** Resulting Value: 143.04 Conclusion: Good fidelity **Evaluation Method 2: PSNR** Resulting Value: 40.04 Conclusion: Good fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.69 Chanel[1]: 0.75 Chanel[2]: 0.70 Mean: 0.71 Conclusion: Good Fidelity		

Threshold: 125 Position: multiple corners			
Image: lena.jpg (512x512, 94.4 Kb) **Watermark object:** Name: watermark 4.jpg Dimension: 128x128 px, Size: 27.6 Kb **Settings** Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1 Threshold: 125 Position: multiple corners	**Evaluation Method 1: MSE** Resulting Value: 163.04 Conclusion: Bad fidelity **Evaluation Method 2: PSNR** Resulting Value: 40.04 Conclusion: Fair fidelity **Evaluation Method 3: SSIM** Resulting Value: Chanel[0]: 0.60 Chanel[1]: 0.65 Chanel[2]: 0.66 Mean: 0.64 Conclusion: Fair Fidelity		

Appendix C, Table 3: Summary of evaluation results for the capacity

Appendix D (Prototype System)

Structure of Prototype Application

/ --- root

/CMake Files -- build files

/GUI --- GUI of application

/GUI/images -- application images

/GUI/source -- source-code of GUI application

main.cpp

/GUI/source/class ------------------------------------ C++ class files of GUI application

evaluate.cpp

mainwindow.cpp

featurewindow.cpp

pixelwindow.cpp

/GUI/source/headers --------------------------------- C++ header files of GUI application

mainwindow.h

/GUI/source/includes -------------------------------- custom header files

headers.h

/GUI/gui.pro --- configuration file

/images -- input image

/result --- output

/source -- source code of watermarking application

main.cpp

/source/compress -------------------------------------- encoder algorithms

render_jpeg.cpp

/source/embed -- embedding algorithms

embed_watermark_corners_harris.cpp

embed_watermark_corners_lochandaka.cpp

embed_watermark_corners.cpp

embed_watermark_pixel_full_fixed_inblock_position.cpp

embed_watermark_pixel_full_fixed_position.cpp

embed_watermark_pixel_full_vari_inblock_position.cpp

embed_watermark_pixel.cpp

/source/evaluate --- evaluation methods

mse.cpp

psnr.cpp

ssim.cpp

/source/extract -- extraction algorithms

extract_watermark_corners_harris.cpp

extract_watermark_corners_lochandaka.cpp

extract_watermark_corners.cpp

extract_watermark_pixel_full_common_position.cpp

extract_watermark_pixel_full_fixed_inblock_position.cpp

extract_watermark_pixel_full_fixed_position.cpp

extract_watermark_pixel_full_vari_inblock_position.cpp

extract_watermark_pixel.cpp

/source/header --- headers

function_decleration.h

includes.h

typedef.h

/source/operators -- feature detection operators

harris_operator.cpp

lochandaka_operator.cpp

/source/read --- read image data

read_jpeg.cpp

read_image.cpp

read_jpeg.cpp

read_pixel.cpp

/source/res --- mathematical functions

convert_color_space.cpp

draw_circle.cpp

drow_image.cpp

filters.cpp

get_corner_points.cpp

get_image_dimensions.cpp

intensity_diff.cpp

math.cpp

pixel_intensity_boundary_adjust.cpp

read_watermark_meta.cpp

Total Lines of code: 6212

Main Screens of Prototype Application

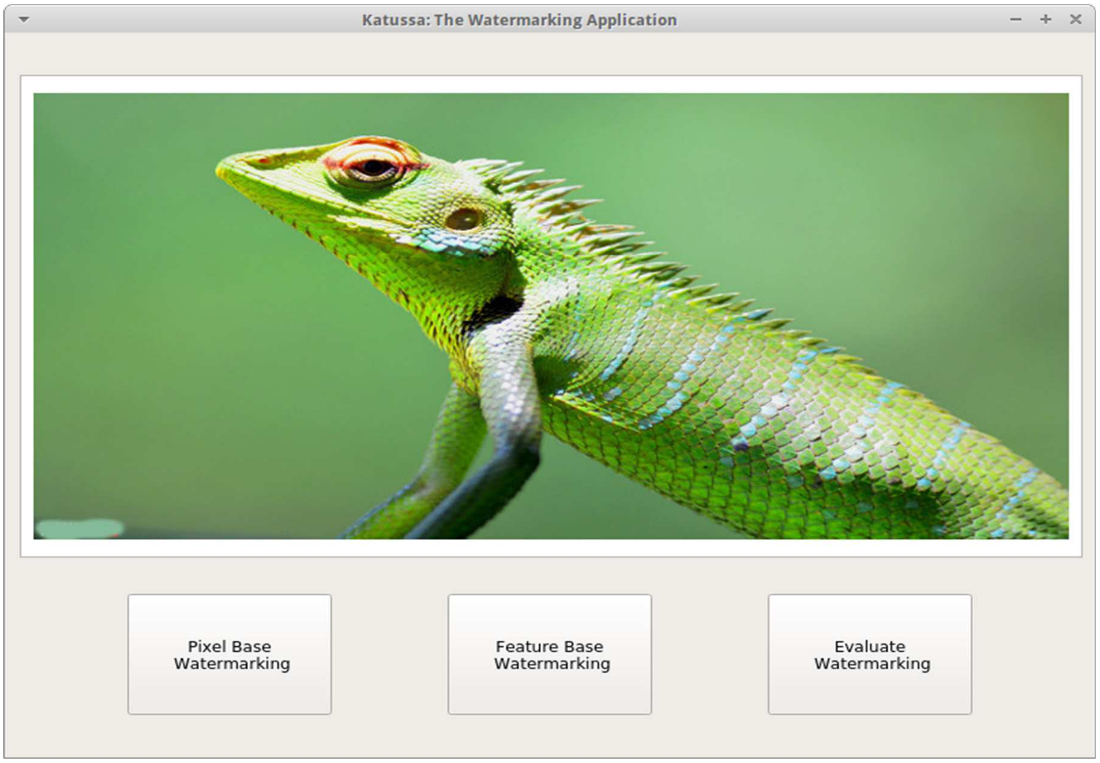
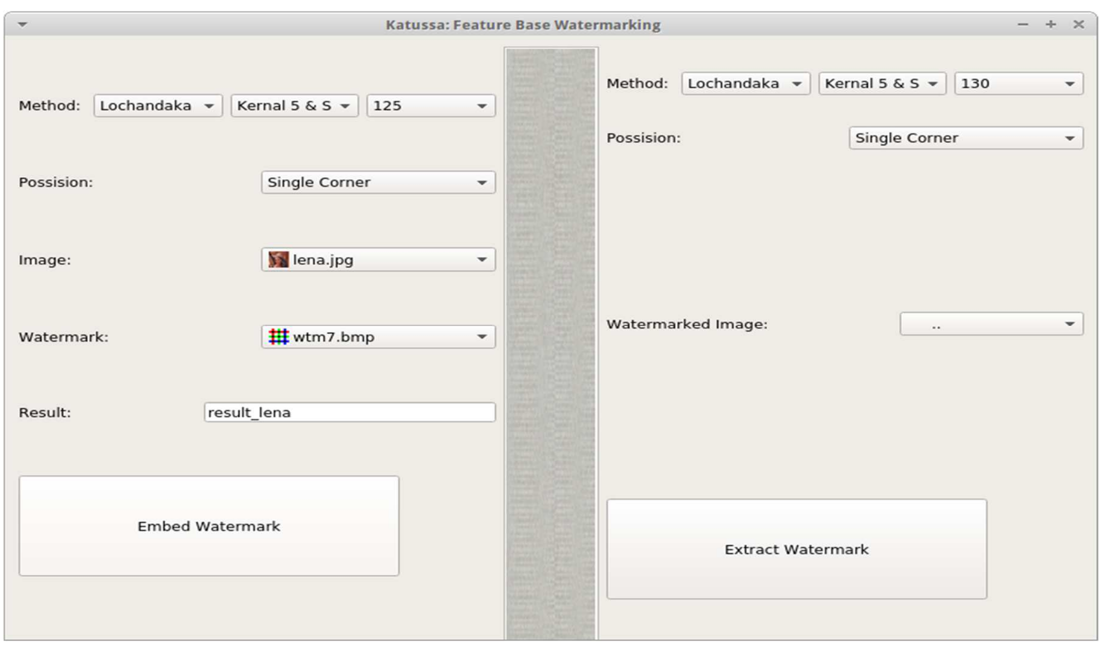


Fig. 1. Appendix D, Figure 1: Main window of prototype system



Appendix D, Figure 2: Watermark embed & extraction window

Appendix D, Figure 2: *Evaluation window*

Source code of novel watermark operators

```
/* Create namespace */
using namespace cv;
using namespace std;

DERIVATIVES katussa_sobel_operator(Mat image);
Mat katussa_gauss_operator_for_lochandaka(Mat IX, int kernal, float sigma);
Mat katussa_lochandaka_response(Mat Gsx, Mat Gsy, Mat Gsxy, float k);
Mat katussa_nonmaximal_suppression_for_lochandaka(Mat response, int threshold);

/**
 * This function implement Harris operator
 * Parameters are image, threshold
 * Return type Mat
 *
 */
Mat katussa_lochandaka_operator(Mat image, float k, int kernal, float sigma, int threshold)
{

	Mat res_img(image.rows, image.cols, CV_8UC1, Scalar(0));
	Mat res_img_maxsupp(image.rows, image.cols, CV_8UC1, Scalar(0));
	Mat Gsx(image.rows, image.cols, CV_32F);
	Mat Gsy(image.rows, image.cols, CV_32F);
	Mat Gsxy(image.rows, image.cols, CV_32F);
	Mat response(image.rows, image.cols, CV_32F);
	DERIVATIVES derivatives;

	// #### (1) Compute X and Y derivatives and compute products of derivatives
```

```
	derivatives = katussa_sobel_operator(image);

	// #### (2) Appy Gaussian operaator to Ix, Iy, Ixy
	Gsx = katussa_gauss_operator_for_lochandaka(derivatives.Ix, kernal,  sigma);
	Gsy = katussa_gauss_operator_for_lochandaka(derivatives.Iy, kernal, sigma);
	Gsxy = katussa_gauss_operator_for_lochandaka(derivatives.Ixy, kernal, sigma);

	// #### (3) Creare matrix for each pixel(x,y) and compute response
	response = katussa_lochandaka_response(Gsx, Gsy, Gsxy, k);

	// #### (4) Non Maximum suppression of response
	res_img_maxsupp = katussa_nonmaximal_suppression_for_lochandaka(response, threshold);

	res_img = res_img_maxsupp;
	return res_img;
}

/**
 * This function implement sobel operator
 * Parameters is Mat image
 * Return type DERIVATIVES
 *
 */
DERIVATIVES katussa_sobel_operator(Mat image)
{

	float derivative_Sx;
	float derivative_Sy;
	float DSx_2 = 0, DSy_2 = 0, DSxy = 0;

	Mat Ix(image.rows, image.cols, CV_32F);
	//Mat Ix(image.rows, image.cols, CV_8UC1, Scalar(0));
	Mat Iy(image.rows, image.cols, CV_32F);
	Mat Ixy(image.rows, image.cols, CV_32F);
	DERIVATIVES sobel_derivatives;

	// Sobel matrix in X direction
	int Sx[3][3] = {
		{ -1, 0, 1 },
		{ -2, 0, 2 },
		{ -1, 0, 1 }
	};

	// Sobel matrix in Y direction
	/*int Sy[3][3] = {
		{ 1, 2, 1 },
		{ 0, 0, 0 },
		{ -1, -2, -1 }
	};*/
	int Sy[3][3] = {
		{ -1, -2, -1 },
		{ 0, 0, 0 },
		{ 1, 2, 1 }
	};

	for (int y=1; y<image.rows-1; y++)
	{
		for (int x=1; x<image.cols-1; x++)
		{
```

```
                                // Compute X derivatives
                                derivative_Sx = 0.0;
                                for (int Sx_y = 0; Sx_y <= 2; Sx_y++)
                                {
                                        for (int Sx_x = 0; Sx_x <= 2; Sx_x++)
                                        {
                                                derivative_Sx += image.at<uchar>(y + (Sx_y-1), x + (Sx_x-1))
* Sx[Sx_y][Sx_x];
                                        }
                                }
                                derivative_Sx = abs(derivative_Sx);
                                //DSx_2 = derivative_Sx*derivative_Sx;
                                Ix.at<uchar>(y, x) = derivative_Sx;
                                //printf("%d ", Ix.at<uchar>(y, x));

                                // Compute Y derivatives
                                derivative_Sy = 0.0;
                                for (int Sy_y = 0; Sy_y <= 2; Sy_y++)
                                {
                                        for (int Sy_x = 0; Sy_x <= 2; Sy_x++)
                                        {
                                                derivative_Sy += image.at<uchar>(y + (Sy_y-1), x + (Sy_x-1))
* Sy[Sy_y][Sy_x];
                                        }
                                }
                                //printf("%.2f ", derivative_Sy);
                                derivative_Sy = abs(derivative_Sy);
                                //DSx_2 = derivative_Sy*derivative_Sy;
                                Iy.at<uchar>(y, x) = derivative_Sy;
                                //printf("%d ", Iy.at<uchar>(y, x));

                                DSxy = derivative_Sx*derivative_Sy;
                                Ixy.at<float>(y, x) = DSxy;

                        }
                }

        sobel_derivatives.Ix = Ix;
        sobel_derivatives.Iy = Iy;
        sobel_derivatives.Ixy = Ixy;

        return sobel_derivatives;
}


/**
 * This function implement gauss operator
 * Parameters is Mat sobel_derivatives
 * Return type Mat
 *
 */
Mat katussa_gauss_operator_for_lochandaka(Mat IX, int kernal, float sigma)
{

        Mat Gs_Mat(IX.rows, IX.cols, CV_32F);
        //int Gs[7][7];
        //float Gs_multiplier;
        float Gs_Ivalue;
```

```
	if(kernal==5 && sigma==1){
		int Gs[5][5] = {
			{ 1, 4, 7, 4, 1 },
			{ 4, 16, 26, 16, 4 },
			{ 7, 26, 41, 26, 7 },
			{ 4, 16, 26, 16, 4 },
			{ 1, 4, 7, 4, 1 }
		};
		float Gs_multiplier = 1/273;

		for (int y=2; y<IX.rows-2; y++)
		{
			for (int x=2; x<IX.cols-2; x++)
			{

				Gs_Ivalue = 0.0;
				for (int Gs_y = 0; Gs_y <= 4; Gs_y++)
				{
					for (int Gs_x = 0; Gs_x <= 4; Gs_x++)
					{
						Gs_Ivalue += IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2)) * Gs[Gs_y][Gs_x];
						//Gs_Ivalue = IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2));
						//Gs_Ivalue2 += Gs_Ivalue * Gs[Gs_y][Gs_x];
					}
				}
				Gs_Mat.at<float>(y, x) = Gs_Ivalue*Gs_multiplier;

			}
		}
	}

	if(kernal==5 && sigma==2){
		int Gs[5][5] = {
			{ 2, 7, 12, 7, 2 },
			{ 7, 31, 52, 31, 7 },
			{ 12, 52, 127, 52, 12 },
			{ 7, 31, 52, 31, 7 },
			{ 2, 7, 12, 7, 2 }
		};
		float Gs_multiplier = 1/571;

		for (int y=2; y<IX.rows-2; y++)
		{
			for (int x=2; x<IX.cols-2; x++)
			{

				Gs_Ivalue = 0.0;
				for (int Gs_y = 0; Gs_y <= 4; Gs_y++)
				{
					for (int Gs_x = 0; Gs_x <= 4; Gs_x++)
					{
						Gs_Ivalue += IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2)) * Gs[Gs_y][Gs_x];
						//Gs_Ivalue = IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2));
						//Gs_Ivalue2 += Gs_Ivalue * Gs[Gs_y][Gs_x];
					}
```

```
                        }
                        Gs_Mat.at<float>(y, x) = Gs_Ivalue*Gs_multiplier;

                }
        }
}

if(kernal==5 && sigma==1.5){
        int Gs[5][5] = {
                { 2, 4, 5, 4, 2 },
                { 4, 9, 12, 9, 4 },
                { 5, 12, 15, 12, 5 },
                { 4, 9, 12, 9, 4 },
                { 2, 4, 5, 4, 2 }
        };
        float Gs_multiplier = 1/159;

        for (int y=2; y<IX.rows-2; y++)
        {
                for (int x=2; x<IX.cols-2; x++)
                {

                        Gs_Ivalue = 0.0;
                        for (int Gs_y = 0; Gs_y <= 4; Gs_y++)
                        {
                                for (int Gs_x = 0; Gs_x <= 4; Gs_x++)
                                {
                                        Gs_Ivalue += IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2)) * Gs[Gs_y][Gs_x];
                                        //Gs_Ivalue = IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2));
                                        //Gs_Ivalue2 += Gs_Ivalue * Gs[Gs_y][Gs_x];
                                }
                        }
                        Gs_Mat.at<float>(y, x) = Gs_Ivalue*Gs_multiplier;

                }
        }
}

if(kernal==7 && sigma==1){

        int Gs[7][7] = {
                { 1, 4, 7, 4, 1 },
                { 4, 16, 26, 16, 4 },
                { 7, 26, 41, 26, 7 },
                { 4, 16, 26, 16, 4 },
                { 1, 4, 7, 4, 1 }
        };
        float Gs_multiplier = 1/273;

        for (int y=2; y<IX.rows-2; y++)
        {
                for (int x=2; x<IX.cols-2; x++)
                {

                        Gs_Ivalue = 0.0;
                        for (int Gs_y = 0; Gs_y <= 4; Gs_y++)
                        {
```

```
                                        for (int Gs_x = 0; Gs_x <= 4; Gs_x++)
                                        {
                                                Gs_Ivalue += IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2)) * Gs[Gs_y][Gs_x];
                                                //Gs_Ivalue = IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2));
                                                //Gs_Ivalue2 += Gs_Ivalue * Gs[Gs_y][Gs_x];
                                        }
                                }
                                Gs_Mat.at<float>(y, x) = Gs_Ivalue*Gs_multiplier;

                        }
                }
        }

        return Gs_Mat;



}

/**
 * This function implement gauss operator
 * Parameters are Mat Gsx, Mat Gsy, Mat Gsxy
 * Return type Mat
 *
 */
Mat katussa_lochandaka_response(Mat Gsx, Mat Gsy, Mat Gsxy, float k)
{

        Mat response(Gsx.rows, Gsx.cols, CV_32F);
        float a11, a12, a21, a22;
        float det;
        float trace;

        for (int y=0; y<Gsx.rows; y++)
        {
                for (int x=0; x<Gsx.cols; x++)
                {

                        a11 = Gsx.at<float>(y, x) * Gsx.at<float>(y, x);
                        a22 = Gsy.at<float>(y, x) * Gsy.at<float>(y, x);
                        a12 = Gsxy.at<float>(y, x);
                        a21 = Gsxy.at<float>(y, x);

                        det = (a11*a22) - (a12*a21);
                        trace = a11 + a22;

                        //printf("%f ", det);
                        response.at<float>(y,x) = abs( (det) - (k * (trace*trace)) ); //abs( Gsx.at<float>(y,
x)*Gsy.at<float>(y, x) );
                        //printf("%f ", response.at<float>(y,x));

                }
        }

        return response;
}
```

```
Mat katussa_nonmaximal_suppression_for_lochandaka(Mat response, int threshold)
{
	Mat res_img_maxsupp(response.rows, response.cols, CV_8UC1, Scalar(0));
	int value;

	for (int y=1; y<response.rows; y++)
	{
		for (int x=1; x<response.cols; x++)
		{
			//printf("%d,%d-%.2f # ",y, x, res_img_maxsupp.at<float>(y, x));
			//printf("%d,%d-%d \n ",y, x, res_img_maxsupp.at<uchar>(y, x));
			if(response.at<uchar>(y, x) < threshold)
			{
				value = 0;
			}
			else
			{
				value = 255;
			}
			res_img_maxsupp.at<uchar>(y, x) = value;
			//res_img_maxsupp.at<uchar>(y, x) = response.at<uchar>(y, x);
		}
	}

	return res_img_maxsupp;
}
```

Source code of watermark generator

```
int katussa_intensity_diff_value(int intensity_original, int intensity_watermark)
{
	int intensity_diff;
	char *binary;
	binary = (char*)malloc(32+1);

	// Check intensity is < 240.
	if(intensity_original>1 && intensity_original<248)
	{
		// Compeire intensity values of original image and watermark.
		if(intensity_original>=intensity_watermark)
		{

			// LSB of difference is `0` add `1` make odd difference.
			// Difference odd means intensity of orginal image > intensity of watermark.
			binary = katussa_decimal_to_binary(sqrt(intensity_original-
intensity_watermark));
			if(atoi(&binary[31])==0)
			{
				intensity_diff =
katussa_binary_addition(katussa_binary_to_decimal(atoi(binary)),1);

			}
			else{
				intensity_diff =
katussa_binary_addition(katussa_binary_to_decimal(atoi(binary)),0);
```

```
			}

		}
		else
		{

			// LSB of difference is `1` add `1` make even difference.
			// Difference even means intensity of watermark > intensity of original image
			binary = katussa_decimal_to_binary(sqrt(intensity_watermark-
intensity_original));
			if(atoi(&binary[31])==0)
			{
				intensity_diff =
katussa_binary_addition(katussa_binary_to_decimal(atoi(binary)),0);

			}
			else{
				intensity_diff =
katussa_binary_addition(katussa_binary_to_decimal(atoi(binary)),1);
			}

		}
	}

	//printf("%s - %s- %d,", binary, &binary[31], intensity_diff);
	free(binary);

	return intensity_diff;
}
```

Source code of embedder

```
/**
 * This function embed a watermark into original image
 * Arguments: int orgimg_width, int orgimg_height (Dimentions of original image)
 * Arguments: int wtmimg_width, int wtmimg_height (Dimentions of watermark image)
 * Arguments: IMG_MATRIX *pixeli_original_image (Pixel values of original image)
 * Arguments: IMG_MATRIX *pixeli_watermark_image (Pixel values of watermark image)
 * This function call several algorithms
 * Return type IMG_MATRIX
 *
*/
IMG_MATRIX *katussa_embed_watermark_corners_lochandaka_single(char *watermark_meta_file_name,
IMG_MATRIX_GRAY *corner_point,

							int orgimg_width, int orgimg_height,

							int wtmimg_width, int wtmimg_height,

							IMG_MATRIX *pixeli_original_image,
IMG_MATRIX *pixeli_watermark_image)
{
```

```
	int index = orgimg_width*orgimg_height*41;
	IMG_MATRIX *img_matrix = NULL;
  img_matrix = (IMG_MATRIX*) malloc(index);

	FILE *watermark_meta_file;
	FILE *watermark_info_file;
	char *watermark_info_file_name;

	// Create watermark meta file
	watermark_meta_file = fopen(watermark_meta_file_name, "w+");

	// Declare the local variables for embed algorithm
	long int total_pixel_image = orgimg_width*orgimg_height;
	int total_pixel_watermark = wtmimg_width*wtmimg_height;
	int x, y;
	int intensity_diff_r, intensity_diff_g, intensity_diff_b;
	//int watermark_embed_index = (total_pixel_image/total_pixel_watermark)-(wtmimg_width/2);

	long int count = 0;
	for(long int i=0; i<total_pixel_image; i++)
	{
		x = i%orgimg_width;
		y = i/orgimg_width;

		if( (y>=corner_point->y && y<corner_point->y+wtmimg_width) && (x>=corner_point-
>x && x<corner_point->x+wtmimg_height) )
		{

			for(int j=0; j<total_pixel_watermark; j++)
			{
				if(count==j)
				{
					intensity_diff_r =
katussa_intensity_diff_value(pixeli_original_image[i].intensity_r, pixeli_watermark_image[j].intensity_r);
					intensity_diff_g =
katussa_intensity_diff_value(pixeli_original_image[i].intensity_g, pixeli_watermark_image[j].intensity_g);
					intensity_diff_b =
katussa_intensity_diff_value(pixeli_original_image[i].intensity_b, pixeli_watermark_image[j].intensity_b);

					img_matrix[i].x = x;
					img_matrix[i].y = y;
					img_matrix[i].intensity_r = pixeli_original_image[i].intensity_r
+ intensity_diff_r;
					img_matrix[i].intensity_g =
pixeli_original_image[i].intensity_g + intensity_diff_g;
					img_matrix[i].intensity_b =
pixeli_original_image[i].intensity_b + intensity_diff_b;

					fprintf(watermark_meta_file, "%d, %d, %d, %d\n", j,
intensity_diff_r, intensity_diff_g, intensity_diff_b);
				}
			}
			count++;
		}
		else
		{

			img_matrix[i].x = x;
			img_matrix[i].y = y;
```

```
                        img_matrix[i].intensity_r = pixeli_original_image[i].intensity_r;
                        img_matrix[i].intensity_g = pixeli_original_image[i].intensity_g;
                        img_matrix[i].intensity_b = pixeli_original_image[i].intensity_b;

                }

        }

        fclose(watermark_meta_file);

        // Create watermark info file
        watermark_info_file_name = (char*)malloc(100);
        strcpy(watermark_info_file_name,  watermark_meta_file_name);
        strcat(watermark_info_file_name, ".info");
        watermark_info_file = fopen(watermark_info_file_name, "w+");
        fprintf(watermark_info_file, "%s, %d, %d, %d, %d\n",
                                        watermark_meta_file_name, orgimg_width, orgimg_height,
wtmimg_width, wtmimg_height);

        return img_matrix;
}
```

Source code of extractor

```
/**
 * This function extract a watermark from watermarked image & watermark meta file
 * Arguments: char* extract method
 * Arguments: char* watermark_meta_file_name
 * Arguments: int orgimg_width, int orgimg_height (Dimentions of original image)
 * Arguments: IMG_MATRIX *pixeli_watermarked_image (Pixel values of watermarked image)
 * Arguments: IMG_MATRIX *pixeli_watermark_meta (Pixel values of watermark meta file)
 * This function call several algorithms
 * Return type IMG_MATRIX
 *
*/
IMG_MATRIX *katussa_extract_watermark_corners_single(IMG_MATRIX_GRAY *corner_point,


                                                int watermarked_width, int watermarked_height,


                                                int wtmimg_width, int wtmimg_height,


                                                IMG_MATRIX *pixeli_watermarked_image, Mat
watermarked_image,


                                                char *watermark_mata_file_name)
{

        int index = wtmimg_width*wtmimg_height*41;
        IMG_MATRIX *img_matrix = NULL;
  img_matrix = (IMG_MATRIX*) malloc(index);

        long int total_pixel_image = watermarked_width*watermarked_height;
```

```
            int total_pixel_watermark = wtmimg_width*wtmimg_height;
            int x_wtm, y_wtm, x_wtmed, y_wtmed; // dimention of watermark image
            int intensity_orgi_r=0, intensity_orgi_g=0, intensity_orgi_b=0;
            int intensity_extrctwtm_r=0, intensity_extrctwtm_g=0, intensity_extrctwtm_b=0;

            // Declare image metrix and allocate memory
            int  index_wtm= total_pixel_watermark*36;
            IMG_DATA *wtm_data = NULL;
  wtm_data = (IMG_DATA*) malloc(index_wtm);

            wtm_data = katussa_read_watermark_meta_data(watermark_mata_file_name);
            /*for(long int i=0; i<total_pixel_watermark; i++)
            {
                        printf("%d(%d, %d, %d)# ", wtm_data[i].index, wtm_data[i].intensity_r,
wtm_data[i].intensity_g, wtm_data[i].intensity_b);
            }*/

            long int count = 0;
            for(int i=0; i<total_pixel_watermark; i++)
            {
                        x_wtm = i%wtmimg_width;
                        y_wtm = i/wtmimg_width;
                        //printf("%d,%d# ", img_matrix[i].y, img_matrix[i].y);

                        for(int j=corner_point->y+y_wtm; j<corner_point->y+1+y_wtm; j++)
                        {
                                    for(int k=corner_point->x; k<corner_point->x+48; k++)
                                    {
                                                img_matrix[i].y = y_wtm;
                                                img_matrix[i].x = x_wtm;
                                                //printf("%d,%d# ", j, k);
                                                //printf("%d,%d# ", img_matrix[i].y, img_matrix[i].x);
                                                int b = watermarked_image.at<cv::Vec3b>(j, k)[0];
                                                int g = watermarked_image.at<cv::Vec3b>(j, k)[1];
                                                int r = watermarked_image.at<cv::Vec3b>(j, k)[2];
                                                //printf("(%d,%d)-%d \n", j, k, b);
                                                //img_matrix[i].intensity_r = r;
                                                //img_matrix[i].intensity_g = g;
                                                //img_matrix[i].intensity_b = b;

                                                // If intensity_r odd (if LSB is 1)
                                                if( (wtm_data[count].intensity_r%2)==1 )
                                                {
                                                            intensity_orgi_r = r - wtm_data[count].intensity_r;
                                                            intensity_extrctwtm_r = intensity_orgi_r -
(wtm_data[count].intensity_r * wtm_data[count].intensity_r);
                                                            img_matrix[i].intensity_r =
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_r);
                                                }
                                                // If intensity_g odd (if LSB is 1)
                                                if( (wtm_data[count].intensity_g%2)==1 )
                                                {
                                                            intensity_orgi_g = g - wtm_data[count].intensity_g;
                                                            intensity_extrctwtm_g = intensity_orgi_g -
(wtm_data[count].intensity_g * wtm_data[count].intensity_g);
                                                            img_matrix[j].intensity_g =
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_g);
                                                }
                                                // If intensity_b odd (if LSB is 1)
```

```
						if( (wtm_data[count].intensity_b%2)==1 )
						{
							intensity_orgi_b = b - wtm_data[count].intensity_b;
							intensity_extrctwtm_b = intensity_orgi_b - 
(wtm_data[count].intensity_b * wtm_data[count].intensity_b);
							img_matrix[j].intensity_b = 
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_b);
						}

						// If intensity_r odd (if LSB is 0)
						if( (wtm_data[count].intensity_r%2)==0 )
						{
							intensity_orgi_r = r - wtm_data[count].intensity_r;
							intensity_extrctwtm_r = intensity_orgi_r + 
(wtm_data[count].intensity_r * wtm_data[count].intensity_r);
							img_matrix[i].intensity_r = 
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_r);
						}
						// If intensity_g odd (if LSB is 0)
						if( (wtm_data[count].intensity_g%2)==0 )
						{
							intensity_orgi_g = g - wtm_data[count].intensity_g;
							intensity_extrctwtm_g = intensity_orgi_g + 
(wtm_data[count].intensity_g * wtm_data[count].intensity_g);
							img_matrix[i].intensity_g = 
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_g);
						}
						// If intensity_b odd (if LSB is 0)
						if( (wtm_data[count].intensity_b%2)==0 )
						{
							intensity_orgi_b = b - wtm_data[j].intensity_b;
							intensity_extrctwtm_b = intensity_orgi_b + 
(wtm_data[count].intensity_b * wtm_data[count].intensity_b);
							img_matrix[i].intensity_b = 
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_b);
						}

					}
			}

			//printf("%ld, ", count);
			count++;
	}

	return img_matrix;
}


/**
 * This function extract a watermark from watermarked image & watermark meta file
 * Arguments: char* extract method
 * Arguments: char* watermark_meta_file_name
 * Arguments: int orgimg_width, int orgimg_height (Dimentions of original image)
 * Arguments: IMG_MATRIX *pixeli_watermarked_image (Pixel values of watermarked image)
 * Arguments: IMG_MATRIX *pixeli_watermark_meta (Pixel values of watermark meta file)
 * This function call several algorithms
 * Return type IMG_MATRIX
 * TODO NEED TO DEVELOPMENT FROMTHE SCRECH
*/
```

```
IMG_MATRIX *katussa_extract_watermark_corners_multiple(IMG_MATRIX_GRAY *corner_point,
                                                       int watermarked_width, int watermarked_height,
                                                       int wtmimg_width, int wtmimg_height,
                                                       IMG_MATRIX *pixeli_watermarked_image, Mat
watermarked_image,
                                                       char *watermark_mata_file_name)
{
        int index = wtmimg_width*wtmimg_height*41;
        IMG_MATRIX *img_matrix = NULL;
  img_matrix = (IMG_MATRIX*) malloc(index);

        long int total_pixel_image = watermarked_width*watermarked_height;
        int total_pixel_watermark = wtmimg_width*wtmimg_height;
        int x_wtm, y_wtm, x_wtmed, y_wtmed; // dimention of watermark image
        int intensity_orgi_r=0, intensity_orgi_g=0, intensity_orgi_b=0;
        int intensity_extrctwtm_r=0, intensity_extrctwtm_g=0, intensity_extrctwtm_b=0;

        // Declare image metrix and allocate memory
        int  index_wtm= total_pixel_watermark*36;
        IMG_DATA *wtm_data = NULL;
  wtm_data = (IMG_DATA*) malloc(index_wtm);

        wtm_data = katussa_read_watermark_meta_data(watermark_mata_file_name);
        /*for(long int i=0; i<total_pixel_watermark; i++)
        {
                printf("%d(%d, %d, %d)# ", wtm_data[i].index, wtm_data[i].intensity_r,
wtm_data[i].intensity_g, wtm_data[i].intensity_b);
        }*/

        long int count = 0;
        for(int i=0; i<total_pixel_watermark; i++)
        {
                x_wtm = i%wtmimg_width;
                y_wtm = i/wtmimg_width;
                //printf("%d,%d# ", img_matrix[i].y, img_matrix[i].y);

                for(int j=corner_point->y+y_wtm; j<corner_point->y+1+y_wtm; j++)
                {
                        for(int k=corner_point->x; k<corner_point->x+48; k++)
                        {
                                img_matrix[i].y = y_wtm;
                                img_matrix[i].x = x_wtm;
                                //printf("%d,%d# ", j, k);
                                //printf("%d,%d# ", img_matrix[i].y, img_matrix[i].x);
                                int b = watermarked_image.at<cv::Vec3b>(j, k)[0];
                                int g = watermarked_image.at<cv::Vec3b>(j, k)[1];
                                int r = watermarked_image.at<cv::Vec3b>(j, k)[2];
                                //printf("(%d,%d)-%d \n", j, k, b);
                                //img_matrix[i].intensity_r = r;
```

```
//img_matrix[i].intensity_g = g;
//img_matrix[i].intensity_b = b;

// If intensity_r odd (if LSB is 1)
if( (wtm_data[count].intensity_r%2)==1 )
{
	intensity_orgi_r = r - wtm_data[count].intensity_r;
	intensity_extrctwtm_r = intensity_orgi_r -
(wtm_data[count].intensity_r * wtm_data[count].intensity_r);
	img_matrix[i].intensity_r =
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_r);
}
// If intensity_g odd (if LSB is 1)
if( (wtm_data[count].intensity_g%2)==1 )
{
	intensity_orgi_g = g - wtm_data[count].intensity_g;
	intensity_extrctwtm_g = intensity_orgi_g -
(wtm_data[count].intensity_g * wtm_data[count].intensity_g);
	img_matrix[j].intensity_g =
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_g);
}
// If intensity_b odd (if LSB is 1)
if( (wtm_data[count].intensity_b%2)==1 )
{
	intensity_orgi_b = b - wtm_data[count].intensity_b;
	intensity_extrctwtm_b = intensity_orgi_b -
(wtm_data[count].intensity_b * wtm_data[count].intensity_b);
	img_matrix[j].intensity_b =
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_b);
}

// If intensity_r odd (if LSB is 0)
if( (wtm_data[count].intensity_r%2)==0 )
{
	intensity_orgi_r = r - wtm_data[count].intensity_r;
	intensity_extrctwtm_r = intensity_orgi_r +
(wtm_data[count].intensity_r * wtm_data[count].intensity_r);
	img_matrix[i].intensity_r =
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_r);
}
// If intensity_g odd (if LSB is 0)
if( (wtm_data[count].intensity_g%2)==0 )
{
	intensity_orgi_g = g - wtm_data[count].intensity_g;
	intensity_extrctwtm_g = intensity_orgi_g +
(wtm_data[count].intensity_g * wtm_data[count].intensity_g);
	img_matrix[i].intensity_g =
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_g);
}
// If intensity_b odd (if LSB is 0)
if( (wtm_data[count].intensity_b%2)==0 )
{
	intensity_orgi_b = b - wtm_data[j].intensity_b;
	intensity_extrctwtm_b = intensity_orgi_b +
(wtm_data[count].intensity_b * wtm_data[count].intensity_b);
	img_matrix[i].intensity_b =
katussa_pixel_intensity_boundary_adjust(intensity_extrctwtm_b);
}
```

```
				}
			}

			//printf("%ld, ", count);
			count++;
		}

		return img_matrix;
}
```